TP Guidé : Les Graphes

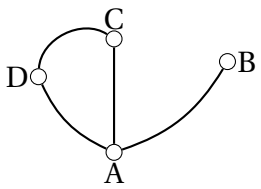
10 novembre 2021

I. Définitions et Premières Propriétés

Définition .1.
Un graphe G est la donnée d'un ensemble de nœuds noté V et d'un ensemble d'arêtes reliant ces nœuds noté E. On écrit G = (V, E)

R Les notations V et E viennent de l'anglais. V signifie Vertex qui veut dire Sommet.
E signifie Edge qui veut dire arête.

▪ **Exemple .1.**
Le graphe G = (V, E) avec V = {A, B, C, D} et E = {[A, B], [A, C], [A, D], [D, C]} ressemble à la figure suivante.



▪

▪ **Exercice .1.**

1. Faire une classe Noeud avec pour attribut Nom.
2. Faire une classe Arête avec pour attributs Noeud1 et Noeud2
3. Faire une classe Graphe avec pour attributs : une liste comportant des objets de type Noeud et une liste comportant des objets de type arête.
4. Construire 2 exemples : G = ([1, 2, 3], ([1, 2], [1, 3])) et
   GG = (([“A”, “B”, “C”]), ([“A”, “C”], [“B”, “C”]))

▪

Il existe un outil mathématique utile pour manipuler ces graphes (et réaliser des modèles). Cet outil est la matrice d'adjacence. Il permet en un seul tableau de nombre de se faire une représentation du graphe (une matrice est un tableau de nombre).

Définition .2.
Soit G = (V, E) un graphe, on peut construire une matrice telle que les entrées de la matrice sont les noeuds et à la case (i, j) on met
- 1 si il y a une arête entre i et j
- 0 sinon

R Dans certains cas, on définit la matrice d'adjacence en mettant à la case (i, j) le nombre d'arêtes reliant i et j.

▪ **Exemple .2.**
Dans le graphe .1, la matrice d'adjacence est :

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

▪

▪ **Exercice .2.**

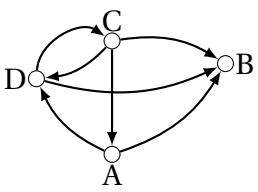
1. Ajouter à votre classe Graphe un attribut adjacence et faîtes une méthode qui calcule la matrice d'adjacence d'un graphe.
2. Donner sur feuille 4 exemples de Graphes. Donner sur feuille leur matrice d'adjacence respective. Vérifier que ces matrices sont bien symétriques par rapport à la diagonale descendante.
3. Faite 4 objets Graphes correspondants aux 4 graphes que vous avez pris en exemple. Utilisez la méthode pour calculer leur matrice d'adjacence. Vérifiez.

▪

II. Graphes Orientés

Définition .3.
Un graphe orienté est la donnée de :
- un ensemble de noeuds noté V
- un ensemble d'arêtes dirigées (une arête ne relie pas A ET B mais elle relie A vers B (et pas forcément le contraire)) noté E.

▪ **Exemple .3.**
Le graphe G = ((A, B, C, D), (A → B, A → D, D → C, C → D, C → B, D → B)) est représenté par :



■

Définition .4.
La matrice d'adjacence d'un graphe orienté G = (V, E) est telle que les entrées de la matrice sont les noeuds et à la case (i, j) on met
— 1 si il y a une arête de i vers j
— 0 sinon

■ **Exemple .4.**
La matrice d'adjacence du graphe G de l'exemple .3 est la suivante.

$$\begin{array}{c} \\ \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \end{array} \begin{array}{c} \begin{array}{cccc} \text{A} & \text{B} & \text{C} & \text{D} \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

■

Propriété .1.
La matrice d'adjacence d'un graphe non orienté est symétrique (par rapport à la première diagonale). Celle d'un graphe orienté ne l'est pas nécessairement.

R Un graphe non orienté est un cas particulier de graphe orienté. Dans la suite, on ne considèrera que des graphes orientés.

■ **Exercice .3.**
Adapter la classe graphe pour qu'elle puisse convenir à des graphes orienté et non orienté. Ajouter un attribut booléen orientation qui vaut *True* si le graphe est orienté et *False* si le graphe est non orienté. ■

III. Voisins et Chemins

Définition .5.
Soit G = (V, E) un graphe, soit $x \in$ V un noeud. Les voisins de x sont les noeuds qui sont liés à x par une arête (de x vers le voisin).

■ **Exemple .5.**
Dans l'exemple .3, les voisins de A sont B et D. C n'est pas un voisin de A car il n'y a pas d'arête qui va de A vers C. ■

■ **Exercice .4.**
Ajouter à la classe Noeud un attribut liste : voisins. Ajouter une méthode FaireVoisin à graphe qui va mettre à jour les attributs voisins de tous les noeuds du graphe. ■

Définition .6.
Soit G = (V, E) un graphe. Un chemin de G reliant le noeud $x \in$ V à $y \in$ V est l'en-

semble des noeuds reliés par une arête menant de x à y.

▪ **Exemple .6.**
Dans l'exemple .3, il y a trois chemins de C à B : le chemin [C, D, B], [C, A, B], [C, B]. ▪

IV. Modèle de Watts : Graphes Small World

R Un graphe Small World est un graphe qui apparaît naturellement dans les réseaux sociaux (qu'ils soient numériques ou non). Nous allons apprendre à en générer un.

▪ **Exercice .5.**
Soit n un entier fixé (vous prendrez $n = 20$) et $p \in]0,1[$ (on prendra $p = 0.3$)

1. Générer un graphe avec n noeuds et une arête reliant chaque paire de noeud.
2. Compter le nombre d'arêtes puis supprimer chaque arête avec une probabilité de p.
3. À l'aide des librairies networkx (comme nx dans la suite) et matplotlib.pypplot (comme plt dans la suite), représenter le graphe que vous venez d'obtenir.

▪

R

— Pour créer un graphe on fait :

```
G = nx.Graph()
```

— Pour ajouter des noeuds et des arêtes on fait respectivement :

```
G.add_nodes_from(Liste_De_Noeuds)
G.add_edges_from(Liste_d'ar\^etes)
```

— Pour tracer et voir le graphe on fait :

```
nx.draw(G)
plt.show()
```

— Vous trouverez d'autres commandes pour dessiner les graphes dans la documentation de Networkx.
— Les noeuds ne sont pas nécessairement des chaînes de caractères, ils peuvent être n'importe quoi, même des objets.

V. Parcours sur les Graphes

Dans cette section, on va voir deux algorithmes de parcours : le parcours en profondeur et le parcours en largeur.

1. Parcours en profondeur

R [1] L'exploration d'un parcours en profondeur depuis un sommet S fonctionne comme suit. Il poursuit alors un chemin dans le graphe jusqu'à un cul-de-sac ou alors jusqu'à atteindre un sommet déjà visité. Il revient alors sur le dernier sommet où on pouvait suivre un autre chemin puis explore un autre chemin (voir vidéo ci-contre). L'exploration s'arrête quand tous les sommets depuis S ont été visités. Bref, l'exploration progresse à partir d'un sommet S en s'appelant récursivement pour chaque sommet voisin de S.

Le pseudo code est le suivant :

```
1 Fonction parcours_en_profondeur(Graphe G, Sommet S):
2     Marquer S comme visité
3     Pour tous les voisins v de S FAIRE:
4         Si v n'est pas visité ALORS :
5             parcours_en_profondeur(G,v)
```

▪ Exercice .6.

1. Implémenter le parcours en profondeur en python.
2. Proposer une procédure de Test de l'algorithme
3. Le tester sur le graphe Small World de l'exercice .5.

▪

R Si l'on veut faire des opérations sur les noeuds du graphes suivant cet algorithme. Les opérations se font durant la phase de marquage.

2. Parcours en Largeur

R [2] L'algorithme de parcours en largeur (ou BFS, pour Breadth First Search en anglais) permet le parcours d'un graphe ou d'un arbre de la manière suivante : on commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc. L'algorithme de parcours en largeur permet de calculer les distances de tous les nœuds depuis un nœud source dans un graphe non pondéré (orienté ou non orienté).

Le pseudo-code est le suivant :

1. Source : Wikipedia
2. Source : Wikipedia

```
1  Fonction parcours_en_largeur(Graphe G, Sommet S):
2      f ← creer_file()
3      f.enfiler(S)
4      Marquer S
5      Tant que f est non vide FAIRE :
6          s ← défiler(f)
7          afficher(s)
8          pour tout voisin v de s FAIRE:
9              Si v est non marqué:
10                 f.enfiler(v)
11                 marquer v
```

R Si l'on veut faire des opérations sur les noeuds du graphes suivant cet algorithme. Les opérations se font durant la phase d'affichage.

▪ Exercice .7.

1. Implémenter le parcours en largeur en python.
2. Proposer une procédure de Test de l'algorithme
3. Le tester sur le graphe Small World de l'exercice .5.

▪

VI. Détection de Cycle

Définition .7.
Un cycle est un chemin dont les deux extrémités sont identiques.

R Pour détecter un cycle, nous allons réaliser un parcours en profondeur et vérifier qu'aucune arête ne va vers un noeud déjà visité.

Le pseudo code est le suivant :

```
1  Fonction recherche_cycle(Graphe G, Sommet S):
2      Marquer S comme visité
3      Pour tous les voisins v de S FAIRE:
4          Si v n'est pas visité ALORS:
5              recherche_cycle(G,v)
6          SINON:
7              renvoyer Vrai
8      renvoyer Faux
```

▪ Exercice .8.

1. Implémenter la détection de cycle en python.
2. Proposer une procédure de Test de l'algorithme
3. Le tester sur le graphe Small World de l'exercice .5.

▪

VII. Recherche d'un chemin

La recherche d'un chemin est très important dans la pratique. Un exemple récent d'utilisation de ces algorithmes serait la recherche d'un patient 0, ou la recherche du plus court chemin pour un trajet. On ne s'intéressera pas dans la suite à la recherche du plus court chemin.

R Pour cet algorithme, on va légèrement modifier l'algorithme de parcours en Largeur

Le pseudo code est le suivant :

```
1  Fonction recherche_de_chemin(Graphe G, Sommet de D
       épart S, Sommet d'arrivée D):
2      f ← creer_file()
3      f.enfiler(S)
4      marquer S
5      Tant que f est non vide FAIRE :
6          s ← défiler(f)
7          pour tout voisin v de s FAIRE:
8              Si v = D ALORS:
9                  return VRAI
10             Si v est non marqué ALORS:
11                 f.enfiler(v)
12                 marquer v
13     return FAUX
```

▪ Exercice .9.

1. Implémenter la recherche de chemin en python.
2. Proposer une procédure de Test de l'algorithme
3. Le tester sur le graphe Small World de l'exercice .5.

▪